

Classification of natural language based on character frequency

Thomas Kerwin

June 7, 2006

Contents

1	Introduction	1
2	Two methods	2
2.1	k-Nearest neighbor	2
2.2	Hidden Markov model	3
3	Implementation	3
3.1	k-Nearest neighbor	3
3.2	Hidden Markov model	4
4	Results	4
4.1	k-Nearest neighbor	5
4.2	Hidden Markov model	6
4.3	Author dataset	6
5	Conclusion	7

1 Introduction

The problem of text classification is a broad one. Specific tasks that can be put into this area include spam filtering, author identification, topic identification and many others. The task of language identification, although a much easier problem, is still significant. For example, in web searches users typically only want results that include pages that they can read. In that

case, being able to identify the language that pages are written in can increase the quality of the search results. Also, for other types of document classification, identifying the language can be an important first step.

In this paper, I discuss the use of two techniques to perform automatic language classification. The first technique is based on the standard k-nearest neighbor method. The second one uses a hidden markov model. Both achieve very good results. I compare and contrast the results based on unigrams and bigrams of characters and also on the size of the testing sample.

2 Two methods

2.1 k-Nearest neighbor

One common and easy to use technique for classification is k-Nearest neighbor. Like most guided learning algorithms, it consists of a training phase and a testing phase. In the training phase, data points are given in a n-dimensional space. These training data points have labels associated with them that designate their class. In the testing phase, unlabeled data are given and the algorithm generates the list of the k nearest (already classified) data points to the unlabeled point. The algorithm then returns the class of the majority of that list.

In a k-nearest neighbor approach, data need to be first represented as some sort of a feature vector so that they can be given a location in the n-dimensional space. In this case, the data being processed is a string. I experimented with two ways of converting the data strings into feature vectors. The easier of the two ways is to take the frequency distributions of the characters in the text: this is referred to later as the unigram model. The more complicated method is to take frequency distributions of pairs of characters: this is referred to later as the bigram model.

With either of these two methods, two problems arise. One is that of extraneous characters in the text. Characters like / , ; , % , and so on, don't seem likely to give information on the language that a text is written in. There are exceptions, of course. The presence of angled quotation marks («») rather than the English (“ ”) could give evidence for or against various languages. The other problem is of capital letters. To minimize the character set used in processing, I converted the text to lower case after it had been input into the program.

2.2 Hidden Markov model

Another common classification method is the hidden Markov model [5]. This method relies on computing the probabilities going from one state to another state. In this particular problem, we view the text as a temporal series, with each new character having a certain probability of appearing next in the sequence based on the language and what the model has seen previously.

For the hidden Markov model, I ignored a unigram model and went directly to a bigram model. In the bigram hidden Markov model of the data for the text, the likelihood of a character appearing depends only on the character that was seen most recently.

3 Implementation

I implemented both systems entirely in Python [1]. Python is a popular language for linguistic analysis and has definite strengths in text processing. Beyond the capabilities of the standard Python library, I used the `FreqDist` and `ConditionalFreqDist` objects from the free and open source library `nltk` [4]. I used the unicode capabilities of python to account for the wide range of characters found in my different test languages, but I restricted my analysis to texts in the ISO 8859-1 (also known as latin-1) encoding, which covers most Western European languages.

3.1 k-Nearest neighbor

The biggest problem with training the k-Nearest neighbor algorithm is that the full set of characters needed to construct the feature vector is not known until all the data is loaded. This is easily fixed, however, by keeping track of the set of all characters (in the case of the unigram feature vectors) or pairs of characters (in the case of bigram feature vectors). I give pseudocode for the construction of feature vectors for the unigram model in Algorithm 1.

The feature vectors can become quite large. In my tests of five languages, the unigram feature vector length was 51, while the bigram feature vector length was 1185. The length of the feature vector can have dramatic consequences on the speed of the algorithm, since the distance function must be calculated between every test vector and every training vector: the distance function is $O(n)$ where n is the length of the feature vector.

Algorithm 1 Train knn with unigram features

```
UniqueCharacters  $\leftarrow \emptyset$ 
for all lang in Languages do
  for all b blocks of text in language lang do
    append text distribution of b to language list lang
    UniqueCharacters  $\leftarrow$  UniqueCharacters  $\cup$  uniquecharacters(b)
  end for
end for
for all text distribution t in language lists do
  x  $\leftarrow$  []
  for all character c in UniqueCharacters do
    append frequency of c to list x
  end for
  append x in list of feature vectors for language of t
end for
```

3.2 Hidden Markov model

The hidden Markov model implementation didn't require the kind of two pass processing that the k-nearest neighbor model needed. I created a HMM for each language using the conditional distribution of characters based on the previous character. I initially had a problem with whitespace. A large portion of the data files I processed was whitespace. However, whitespace is mainly a formatting issue and does not enter into differences between languages *per se*. So, I count all sequences of whitespace characters as just a single space. As in the k-nearest neighbor model, I simply ignore punctuation and convert all characters to lowercase.

4 Results

For all of these general language results, I used around 500k of text to train each language. The languages I tested are English, Finnish, French, German, Italian, and Middle English. For training and testing of Middle English, my only source was from the works of Geoffrey Chaucer, so results might well be different for a different dialect from that time period. All my data files, with the exception of Chaucer are from the Project Gutenberg website [2], with the generic header stripped out. For speed comparisons, I used a Pentium 4 3.2 GHz system for all tests.

4.1 k-Nearest neighbor

The difference in running time between unigram and bigram tests in the k-nearest neighbor algorithm is significant, as mentioned above. The full battery of tests for blocks of 1000 characters takes a little over forty-two seconds, while the same test using bigrams takes nearly twelve minutes. Profiling confirmed that most of the time spent was in the distance function, even when only returning the square of the distance to eliminate the time taken for the square root.

Accuracy for the bigram model is better than for the unigram model, as you can see in the following charts. The accuracy also increases if more characters are used from the test file to construct one feature vector. Most of the errors in the English and Middle English tests confuse the two languages. The algorithm had much more trouble deciding between those two languages than any other pair.

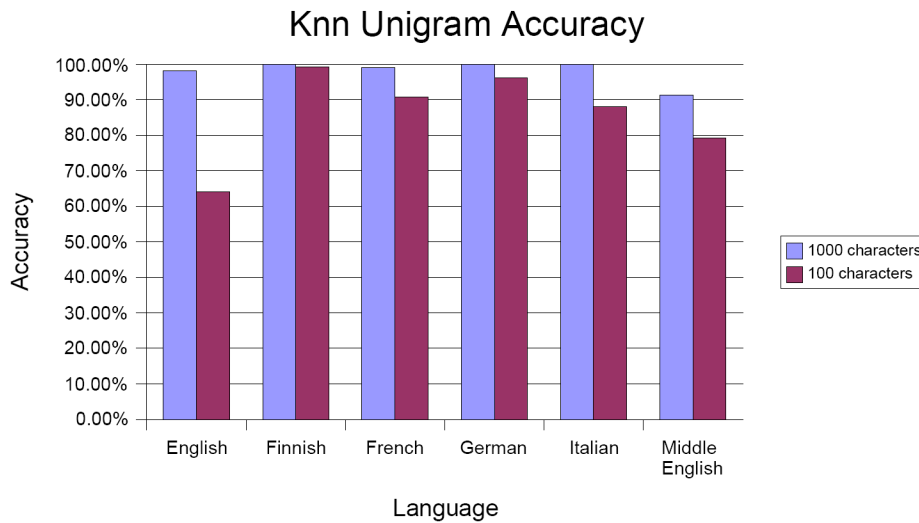


Figure 1: Accuracy for unigram feature vectors with k-nearest neighbor algorithm ($k=10$)

The value of k did not seem to be a major factor in accuracy. Increasing k to 50 resulted in improvement in some languages but a degradation in accuracy for others. The graph is not shown, but increasing k to 100 decreased accuracy for all languages.

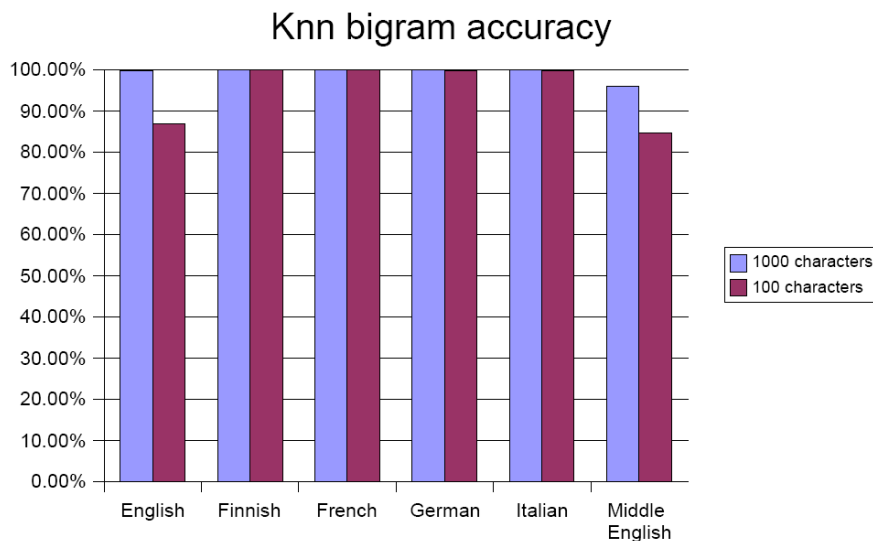


Figure 2: Accuracy for bigram feature vectors with k-nearest neighbor algorithm (k=10)

A strange result I found was that nearly all of the incorrect classifications of Middle English were in a specific file of *A Treatise on the Astrolabe* by Chaucer. The other texts by Chaucer had much greater accuracy than the average shown in the graphs.

4.2 Hidden Markov model

The hidden Markov model had much greater operating performance in terms of speed than even the unigram k-Nearest neighbor model. This was not unexpected, since while in the knn model, test data is compared against all training feature vectors, in the HMM model the test data is only compared to one model per language.

The accuracy of the hidden Markov model was also much greater than the k-Nearest neighbor model. Even with only 100 characters to make a determination, the system had over 90% accuracy for all languages.

4.3 Author dataset

With such high accuracy in the language dataset, I tried the software on some tougher data. I trained both systems on about three megabytes of

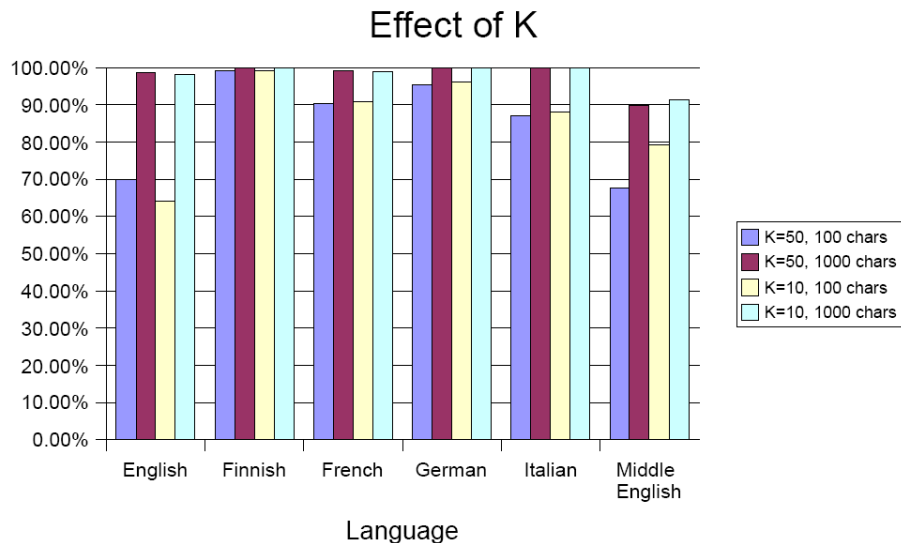


Figure 3: Effect of changing k from 10 to 50. All four tests were performed with the unigram model.

text from Shakespeare and Sir Arthur Conan Doyle. I chose these authors because of the large amount of text I could download by them from the Project Gutenberg website. I tested on a set of around 500 kilobytes text from each of the authors.

The results were good. Surprisingly, the knn unigram-based model increased the accuracy for Shakespeare text when given smaller data blocks to test on, although this is probably an aberration. The HMM-based model did well even when given only 100 characters to make a determination, achieving an accuracy of more than 75% in that case. A similar, although more complicated system was constructed to differentiate between Russian authors by Khmelev [3].

5 Conclusion

For language classification tasks, it seems like, from my experiments, hidden Markov model is a much better choice than k -Nearest neighbor. Accuracy is on par with knn for some tests, but exceeds it in others. Also, performance of the HMM model does not depend on the amount of training data, only on the number of classes. Even with only a couple megabytes of training data,

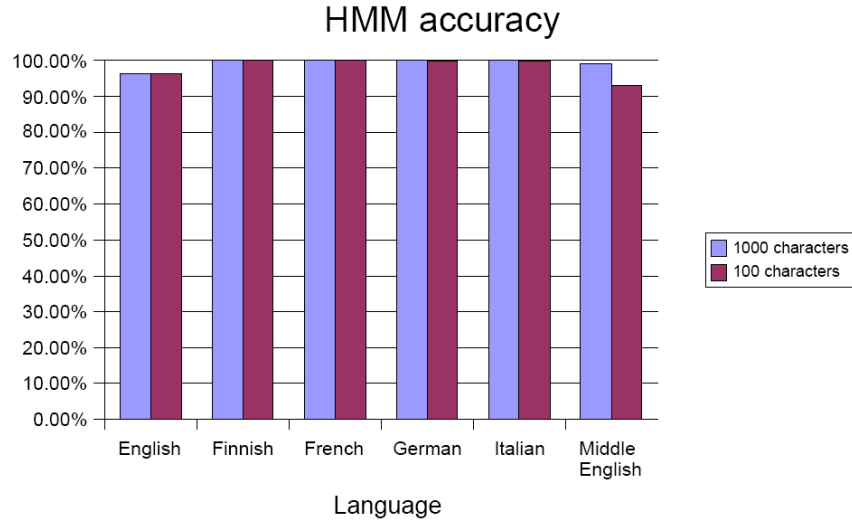


Figure 4: Accuracy for hidden Markov model on language dataset

running the knn model took over ten minutes in some cases, while the HMM model took under a minute.

Although the author classification results are very good, they might very well be due to general language changes in the nearly four centuries between the writings of Shakespeare and Doyle rather than stylistic differences between the two authors. A similar comparison between the writings of two contemporary authors could shed more information on the power of this, very simple, technique for classifying texts.

References

- [1] Python Software Foundation. <http://www.python.org/>, 2006.
- [2] Project Gutenberg. <http://www.gutenberg.org/>, 2006.
- [3] D.V. Khmelev. Disputed authorship resolution through using relative empirical entropy for markov chains of letters in human language texts. *Journal of Quantitative Linguistics*, 7:210–207, 2000.
- [4] E. Loper and S. Bird. Nltk: The natural language toolkit, 2002.

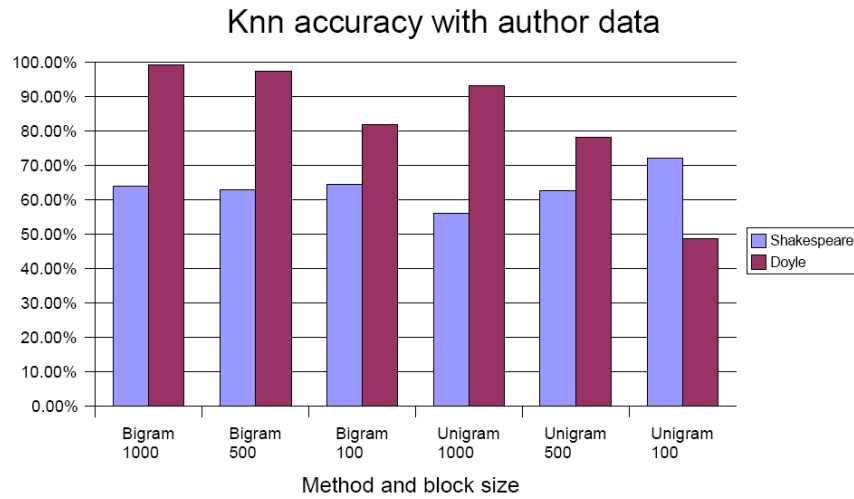


Figure 5: Accuracy for knn model on author dataset

[5] Chris Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. 1999.

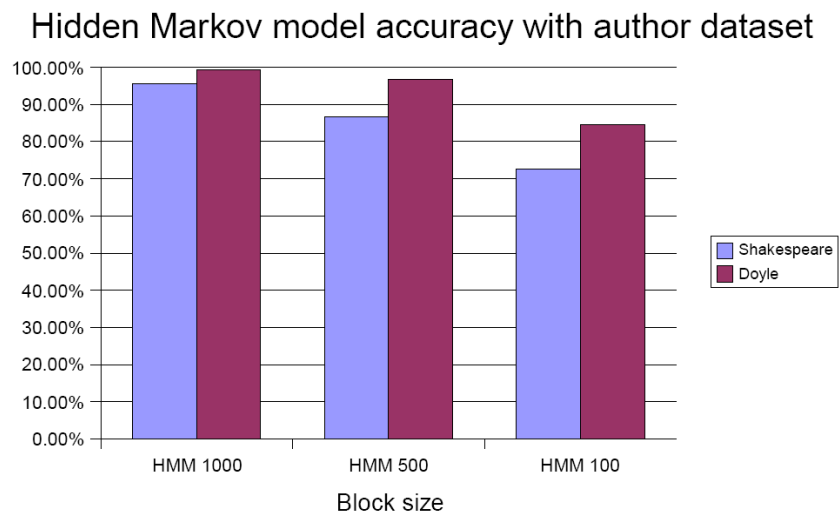


Figure 6: Accuracy for hidden Markov model on author dataset